PATENT ABSTRACTS OF JAPAN

(11)Publication number : 04-100148

(43)Date of publication of application : 02.04.1992

(51)Int.Cl. G06F 9/06
G06F 15/00

(21)Application number : 02-263242

(22)Date of filing : 02.10.1990

(71)Applicant : SUN MICROSYST INC

(72)Inventor : CORBIN JOHN R

(30)Priority

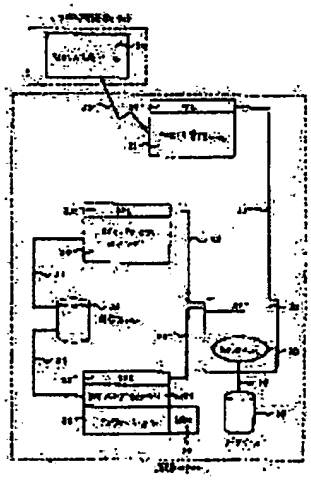
Priority number : 89 415984 Priority date : 02.10.1989 Priority country : US

(54) METHOD FOR PREVENTING UNAUTHORIZED USE OF SOFTWARE INSIDE COMPUTER NETWORK SYSTEM

(57)Abstract:

PURPOSE: To prevent unauthorized use of a software application by confirming a compounded license token means before the certification of access to a license.

CONSTITUTION: When a user desires the execution of the software application, a licensing library 24 performs calling for requesting a license token from a license server 20. The license server 20 detects the proper license token of the software application and transmits this license token to the licensing library 24, and a license access module connected to the licensing library 2 the license token. A routine inside the licensing library coupled to the software application 26 barrows the license and confirms license information before the license token is updated. Thus, the illegal use of the software application is prevented.



BEST AVAILABLE COPY

⑲日本国特許庁(JP)　⑪特許出願公開

⑫公開特許公報(A)　平4−100148

⑤Int. Cl.⁵	識別記号	庁内整理番号
G 06 F 9/06	450 P	7927−5B
15/00	330 B	7218−5L

⑬公開　平成4年(1992)4月2日

審査請求　未請求　請求項の数 2　(全17頁)

⑭発明の名称　コンピュータネットワークシステム内でのソフトウエアの不正使用を防ぐ方法

⑳特　願　平2−263242
㉒出　願　平2(1990)10月2日
優先権主張　㉜1989年10月2日㉝米国(US)㉛415,984

⑫発明者　ジョン・リチャード・コービン　アメリカ合衆国　94086　カリフォルニア州・サニーヴェイル・アメリカ　アヴェニュウ・372
⑪出願人　サン・マイクロシステムズ・インコーポレーテッド　アメリカ合衆国　94043　カリフォルニア州・マウンテンビュー・ガルシア・アヴェニュウ・2550
⑭代理人　弁理士　山川　政樹　外3名

明　細　書

1.発明の名称

コンピュータネットワークシステム内でのソフトウェアの不正使用を防ぐ方法

2.特許請求の範囲

(1)　少なくとも1個のネットワークを構成する複数の端末上で作動すべくライセンスされた複数のソフトウェアアプリケーションを含むコンピュータネットワークの環境であって、前記アプリケーションは前記端末上にあり、特定の端末におけるアプリケーションの使用はライセンスの付与によって認可され、前記ライセンスは前記アプリケーションの存在する前記端末からユーザによって要求されることを特徴とするコンピュータネットワークの環境において、前記アプリケーションの不正使用を防ぐシステムであって、

前記アプリケーションのライセンシング情報を記憶するライセンストークン手段と、

前記端末に接続されて前記アプリケーションと通信を行なうライセンスサーバ手段であって、前記ライセンストークン手段を記憶するデータベースを有し、前記アプリケーションによるライセンスへの要求に応じて前記データベースから前記ライセンストークン手段を検索し、さらに前記ライセンストークン手段を前記アプリケーションへと伝達するライセンスサーバ手段と、

前記端末に接続され、前記ライセンスサーバ手段から伝達されるライセンストークン手段の符号化と復号化とを行なうライセンスアクセス手段であって、前記アプリケーションと一体化され、前記ライセンストークン手段を前記ライセンスサーバ手段から受信するライセンスアクセス手段と、

前記端末に接続され、前記ライセンスへのアクセスが認可される前に前記復号化されたライセンストークン手段を確認するライセンシングライブラリ手段であって、前記アプリケーションと一体化されたライセンシングライブラリ手段と、

からなることを特徴とする前記アプリケーションの不正使用を防ぐシステム。

(2) 少なくとも1個のネットワークを構成する複数の端末上で作動すべくライセンスされた複数のソフトウェアアプリケーションを含むコンピュータネットワークの環境であって、前記アプリケーションは前記端末上にあり、特定の端末におけるアプリケーションの使用はライセンスの付与によって認可され、前記ライセンスは前記アプリケーションの存在する前記端末からユーザによって要求されることを特徴とするコンピュータネットワークの環境において、前記アプリケーションの不正使用を防ぐシステムであって、

前記アプリケーションのライセンシング情報を記憶するライセンストークン手段と、

前記端末に接続されて前記アプリケーションと通信を行なうライセンスサーバ手段であって、前記ライセンストークン手段を記憶するデータベースを有し、前記アプリケーションによるライセンスへの要求に応じて前記データベースから前記ライセンストークン手段を検索し、さらに前記ライセンストークン手段を前記アプリケーションへと伝達するライセンスサーバ手段と、

前記アプリケーションに接続され、前記端末からアクセス可能であり、前記ライセンスサーバ手段から伝達されるライセンストークン手段の符号化と復号化とを行なうライセンスアクセス手段であって、前記アプリケーションと一体化されたライセンスアクセス手段と、

前記アプリケーションに接続され、前記端末からアクセス可能であり、前記ライセンスへのアクセスが許可される前に前記復号化されたライセンストークン手段を確認するライセンシングライブラリ手段であって、前記アプリケーションと一体化されたライセンシングライブラリ手段と、

前記ライセンスサーバ手段と前記ライセンシングライブラリ手段とに接続され、結合ファイルを形成するライセンス結合手段であって、前記結合ファイルは前記ライセンシングライブラリ手段に対して前記ライセンスサーバ手段のうち、いずれの手段が前記アプリケーションにライセンスを付与できるかを通知することを特徴とするライセンス結合手段と、

からなることを特徴とする前記アプリケーションの不正使用を防ぐシステム。

3.発明の詳細な説明

[産業上の利用分野]

本発明は、コンピュータネットワーク環境におけるソフトウェアアプリケーションの不正使用を防ぐ方法に関する。

[従来技術]

コンピュータネットワークとは、一般にリンクやケーブルで相互に接続された複数のマシンまたは端末をいう。コンピュータネットワークは、アクセスに対してオープンであるという特徴を有するため、ソフトウェアを許可なく複製することが可能である。このためソフトウェア開発者のライセンシング収入が減少する結果となっている。従来、ライセンスによってネットワーク全体の実施の許諾を行なう（一般にサイトライセンスと呼ばれる）か、もしくはソフトウェアが実行される各ノードをライセンスする（一般にノードライセンスと呼ばれる）必要があった。ノードとは、コンピュータネットワーク内の各マシン、端末もしくはシステムを指す。ライセンスとは、ソフトウェアの開発者が顧客に対して任意のソフトウェアアプリケーションを特定の様態で使用することを許諾することをいう。

サイトライセンスの場合、特定の場所すなわちネットワーク上の全ユーザに、各々のネットワーク上の位置に関係なくソフトウェアアプリケーションの使用を許可する。均一の使用料を徴収するこの方法は、使用頻度の低いソフトウェアアプリケーションの場合には行き過ぎとなる。ノードライセンスの場合、ソフトウェアアプリケーションをネットワーク上の特定のマシンに結びつけてしまうことになり、またソフトウェアアプリケーションの使用頻度が低ければ採算がとれない。これらの点については米国特許4,888,169に開示されている。またライセンスを付与されたノードで新

たなユーザがソフトウェアアプリケーションを使用しようとする場合、こうしたユーザはライセンスの新規購入を求められることが多い。

サイトライセンスやノードライセンスに代わる概念として並行使用ライセンスがある。並行使用ライセンスでは、任意の時点でソフトウェアアプリケーションの使用を許可されるユーザの月数を、ネットワーク上の位置に係わりなく限定している。ビデオ愛好者がレンタルビデオ店から映画のビデオを借り出すように、ネットワーク上のユーザはソフトウェアアプリケーションを端末から先着順に借り出す。このように、並行使用ライセンスでは、ソフトウェアアプリケーションの実際の使用度に比例して使用料が徴収される。

ネットワーク環境でソフトウェアアプリケーションの並行使用を許可するというライセンシング方法は、現在ハイランド・ソフトウェア社とアポロ・コンピュータ社で採用されている。これに関しては、「並行アクセスライセンシング(Concurrent Access Licensing)」(1988年9月、Vol. 6, No. 9、ユニックス・レビュー(Unix Review)、H. オルソン、P.ルヴァイン(H. Olson and P. Levine)著)で述べられている。一般に、ソフトウェアアプリケーションのライセンスは、ライセンスサーバの制御するデータベースに記憶されている。ライセンスサーバとは、ライセンスを記憶しているプログラムであり、かつライセンスを貸出す前にユーザの資格を確認するプログラムである。不正使用を防ぐために並行使用をライセンスするこれらの方法は、公共/個人キー暗号化方式などによって保護された通信手段に依存している。かかる公共/個人キー暗号化方式によれば、システムの各ユーザは2個のキーを有する。そのうち1個のキーは通常公共のキーであり、他の1個は個人用のキーである。個人キーを使用した個人変換は、公共キーを使用した公共変換に関連付けられているが、個人キーは公共キーから計算によって求めることはできない。これらの点については、D.デニング(D. Denning)著「暗号化技術とデータ保護(Cryptography and Data Security)」(1982年、アディソン-ウェスリー(Addison-Wesley)社刊)に述べられている。暗号キーはライセンスサーバ内に隠されていて、ライセンスのデータベースの暗号化に使用される。高度な技術で設計された公共/個人キー暗号化方式を解読することは難しい。とくにライセンスサーバが安全な環境内にあれば、かかる方式の解読は至難の技である。安全な環境とは、そのアクセスが資格を有するユーザのみに限定されている環境をいう。しかしながら、ライセンスサーバは顧客のサイト、すなわち危険な環境にに設置されることが多い。したがって、ライセンスサーバは熟練したハッカーの解読にさらされることになる。いったん個人キーが解読されると、ライセンスサーバに関するあらゆる重要な情報(ライセンス等)が無許可のまま公開されてしまう。

したがって、本発明の目的は、並行使用ライセンシングの環境におけるソフトウェアの不正使用を防ぐためのより安全性な方法を提供することにある。

[発明の概要]

本発明は、通常ライセンスサーバが行なっていた確認機能とライセンス貸出機能とをソフトウェアアプリケーションに与えるものである。本発明の好ましい実施例は、少なくとも1個のライセンスサーバと少なくとも1個のソフトウェアアプリケーションとを実行する複数の端末を含むコンピュータネットワークからなる。ライセンスサーバは、ソフトウェアアプリケーションのライセンス情報を格納している端末のデータベースを制御する。ライセンス情報はライセンストークンに含まれており、またライセンスサーバが制御するデータベースに記憶されている。ライセンストークンは、特定のビットパターンすなわちパケットであって、アプリケーションソフトウェアの販売業者によって暗号化されている。ソフトウェアアプリケーションは、ライセンシングライブラリを介してライセンスサーバと通信を行なう。ライセンシングライブラリはライブラリルーチンの集合であって、ソフトウェアアプリケーションがライセン

スをライセンスサーバから要求する場合か、もしくは該ライセンスを更新する場合にこれらのライブラリルーチンを使用する。ライセンスアクセスモジュールは、ソフトウェアアプリケーションとライセンシングライブラリとに接続しているプログラムであって、ライセンストークンを販売業者に特定のフォーマットからライセンシングライブラリのフォーマットへと復号化するプログラムである。

ユーザがソフトウェアアプリケーションの実行を希望する場合、ライセンシングライブラリは、ライセンストークンをライセンスサーバから要求するための呼出しを行なう。従来、ライセンスサーバはユーザの資格を確認したのち該要求を許可または却下していたが、本実施例のライセンスサーバは、当該ソフトウェアアプリケーションの正しいライセンストークンを検出してこれをライセンシングライブラリに伝達する。ライセンシングライブラリに接続されたライセンスアクセスモジュールは該ライセンストークンを復号化する。ソフトウェアアプリケーションに結合されたライセンシングライブラリ内のルーチンは、ライセンスを借り出してライセンストークンを更新する前に、ライセンス情報を確認する。ライセンスアクセスモジュールは、更新されたライセンストークンを符号化したのち、該トークンをライセンスサーバへと返送する。

ライセンストークンの確認機能と借り出し機能とがソフトウェアアプリケーションによって実行されるため、ライセンスサーバに代わってソフトウェアアプリケーションが無許可ユーザの解読の対象となる。ライセンスアクセスモジュールにリバースエンジニアリングをかけても、ライセンスサーバの解読より実りが少ない。この理由は、ライセンスアクセスモジュールはライセンスデータベースのごく一部の内容を開示するに過ぎないからである。大部分のハッカーがライセンスアクセスモジュールを解読する頃には、ソフトウェア販売業者は当該ソフトウェアアプリケーションの新バージョンと、これに対応する新たなライセンスアクセスモジュールとを発表しているはずである。このように、本発明は、基本的なコンピュータネットワークを変更することなく、コンピュータネットワーク環境におけるソフトウェアアプリケーションの不正使用を防ぐためのより安全な方法を提供する。

[表記と名称]

以下の詳細な説明では、データビットやコンピュータメモリ内のデータ構造に対して行なわれる演算のアルゴリズムや記号表記を主として使用する。アルゴリズムに基づくこれらの記述や表記は、データ処理分野の有能な技能者が、各々の作業の内容を他の同業者に最も効果的に伝達するために使用する手段である。

以下でいうアルゴリズムとは、通常定義されるように、所望の結果を得るための一貫性のある一連のステップをいう。これらのステップは、物理量の物理的な操作を必要とするステップである。これらの物理量は、必ずしも限定されないが、電気的もしくは磁気的な信号の形態をとることが普通であり、かかる信号に対しては記憶、転送、組合せ、比較その他の操作を行なうことができる。主として慣例上の理由で、これらの信号をビットパターン、値、、要素、記号、文字、データパッケージ等の名称で呼ぶことが便利であることが多い。しかしながら、これらの用語もしくはこれに準ずる用語は、適切な物理量に対応していなければならず、かつ該用語はこれらの数量に付加された便利なラベルに過ぎないことを承知しておく必要がある。

さらに、これらの物理量に対する操作は、加算や比較といった用語で表現され、こうした用語は人間が自分自身で行なう演算に通常関連付けられている。しかしながら、以下で説明する本発明の一部をなす演算では、人間のかかる能力はほとんどの場合不用であり、むしろ好ましくない。演算はマシンが実行するからである。本発明は、汎用ディジタルコンピュータもしくは同様の装置に適用して効果的である。いずれの場合でも、コンピ

ュータを操作する方法と、計算自体の方法とを明確に区別する必要がある。本発明は、電気信号もしくは他の物理的（機械的、化学的など）信号を処理するためにコンピュータを操作することによって、他の所望の物理的信号を生成するステップの方式に関する。

また本発明は、上記の操作を実行する装置に関する。かかる装置は、所望の目的のためのみに形成されるか、もしくは汎用コンピュータからなり、該コンピュータ内に記憶されたコンピュータプログラムによって選択的に起動され、再構成される。本明細書に含まれるアルゴリズムは、特定のコンピュータやその他の装置に限定されるものではない。また各種の汎用マシンを、本明細書記載の命令に準じて記述されたプログラムとともに使用してもよい。所定のステップを実行するための専用の装置を形成することが便利である場合もある。これらの各種マシンの所定の構造は、以下の説明から明瞭になるであろう。

[発明の実施例]

以下の説明は複数の節に分割されている。最初の節では、ライセンスされたソフトウェアプログラムのデータベースをアクセスするための汎用ネットワーク環境を説明する。続く節では、ソフトウェアアプリケーションの不正使用を防ぐ方法の詳細を説明する。

I. 汎用ネットワーク環境

第1図において、コンピュータネットワーク環境は、複数のデータ処理装置10 - 10"（図中では10、10'、10"）からなる。これらのデータ処理装置には、端末、パソコン、ワークステーション、ミニコン、メインフレームコンピュータ、スーパーコンピュータ等が含まれる。本明細書では、本発明のネットワークに接続されたすべてのデータ処理装置を「端末」と称する。これらの端末の製造元は異なるものであって差支えない。また該端末の使用するオペレーティングシステムは、MS-DOS, UNIX, OS/2, MAC OS等さまざまであって差支えない。好ましい端末の例としては、サン・マイクロシステムズ(Sun Microsystems)社（在カリフォルニア州マウンテンビュー(Mountain View)）の製造するマシンがある。各端末は、キーボード11、11'、11"またはマウス12、12'、12"などの入力装置を有する。第1図に示されるように、端末10 - 10"（図中では10、10'、10"）は共通ケーブル13によって接続され、相互にデータの転送を行なう。共通ケーブル13は、同軸ケーブル、光ケーブル、無線チャネルその他のいかなる共有メディアでもよいことは当業者には明白であろう。さらにケーブル13と端末10 - 10"（図中では10、11'、10"）からなるネットワークは、環状結線、星形結線、バス結線など多数の形態のうちいずれをもとることができ、またゲートウェイやブリッジで結合された複数の小形のネットワークの集合を含んでいてもよい。

再び第1図において、14はライセンスサービスを示す。ライセンスサービス14は、ネットワークに接続されたすべての端末が共有する資源である。本実施例では、ライセンスサービス14にはライセンスサーバ15 - 15"（図中では15、15'、15"）とデータベース17 - 17"（図中では17、17'、17"）が含まれる。ここで、nはn以下の値である。ライセンスサーバは、メモリ型記憶機能を有する端末上で実行されるプログラムである。各ライセンスサーバ15（図中では15、15'、15"）は、インタフェース16（図中では16、16'、16"）を介して、端末のメモリに記憶されたデータベース17と通信を行なう。以下で詳しく説明するように、データベース17は、各種のソフトウェアアプリケーションのライセンシング情報を記憶している。これらのソフトウェアアプリケーションは、購入後、コンピュータネットワーク環境で実行するための許可を付与される。ライセンスサーバは、特定の端末での実行に限定されるわけではなく、いずれの端末でも実行可能であり、ユーザがアプリケーションを使用する端末でも実行できる。このように、ネットワークに接続された端末であれば、ユーザがアプリケーションソフトウェアを実行する端末

としてだけでなく、ライセンスサーバとしても機能できる。以下で詳しく説明するように、ライセンスサーバはアプリケーションソフトウェアのライセンスの確認を行なわない。ライセンスサーバの作動形態は受動的であって、記憶機能、ロック機能、ログ機能およびクラッシュ回復機能をアプリケーションソフトウェアに提供している。

第2図は、本発明に基づくネットワークライセンシング方式の構造を示す。この構造には、データベース18、データベースインタフェース19、ライセンスサーバ20、ライセンシングライブラリ24、ライセンスアクセスモジュール27、ライセンス管理ツール21、ライセンスサービスバインダ29、ライセンス生成ツール34が含まれる。

データベース18は、ライセンシング情報とアプリケーション使用データを記憶している。データベース18には、以下の情報を含む複数のレコードが格納されていることが望ましい。

データベースの要素	概要
・一意キーテーブル	他の全テーブル用キー
・販売業者テーブル	販売業者の識別記号(ID)と名前
・製品テーブル	製品の番号と名前
・バージョンテーブル	バージョンの番号と日付
・ライセンステーブル	ライセンス番号、失効日、総数
・ライセンストークンテーブル	符号化されたライセンストークンを格納するテーブル
・単位グループテーブル	任意のグループ内のライセンスの配分
・グループリストテーブル	グループの名前
・使用許可ユーザのテーブル	使用許可ユーザの資格
・現行ライセンス使用テーブル	ライセンスを使用中のアプリケーション

データベースの要素	概要
・ロックテーブル	データベース中のロックされたレコード
・有資格管理者テーブル	管理者のログイン名
・ライセンス操作ログテーブル	管理者のログ情報
・ライセンス使用ログテーブル	要求操作とクライアントログ
・ライセンス待ち行列ログテーブル	ライセンス待ち行列
・アプリケーションメッセージログテーブル	アプリケーションに固有なメッセージ

データベースインタフェース19は、ライセンスサーバ20とデータベース18との間の通信を提供する。これによって複数のユーザがデータベース内の同一のレコードに同時にアクセスすることを防いでいる。かかる同時アクセスが行なわれると、該レコード内のデータが破壊されることがある。このように、アプリケーションの使用時にはロックの所有者のみがロックされたレコードからデータを読み出し、書込むことができる。

ライセンスサーバ20は端末上で作動し、データベース18と、ライセンス管理ツール21、ライセンシングライブラリ24およびライセンスサービスバインダ29とのインタフェースをとる。ライセンスサーバ20は、インタフェース23を介して、ライセンス管理ツール21、ライセンシングライブラリ24およびライセンスサービスバインダ29と通信を行なう。インタフェース23は、遠隔手順呼出し機構であることが望ましい。遠隔手順呼出し機構によれば、ネットワークに接続された1個の装置または端末上で作動するプロセスは、該ネットワークに接続された遠隔装置または遠隔端末から資源またはサービスを要求できる。この点に関しては、A.ビレル、B.ネルソン(A. Birrell and B. Nelson)著「遠隔手順呼出しの実施(Implementing Remote Procedure Calls)」(1984年Vol. 2, No. 1, エーシーエム・トランザクション・オン・コンピ

ュータ・システムズ(ACM Transaction on Computer Systems)記載)に述べられている。

複数のライセンスサーバが複数の端末に常駐することができる。ライセンスサーバ20は、端末のバックグラウンドモードで作動することによって、該端末を使用するユーザからみて透過的であることが望ましい。また以下で説明するように、ライセンスサーバ20は次のような機能を提供する。

(1) ライセンシングライブラリからなされるライセンストークンの要求を処理する機能

(2) いずれのライセンシング単位も使用できない時、データベース18への要求で構成される待ち行列を管理する機能

(3) データベース18への排他的アクセスのロックを生成する機能

(4) データベース18内の情報にアクセスする機能

ライセンシングライブラリ24は、ライブラリルーチンの集合であり、アプリケーション26はこれらのルーチンによってライセンスサーバ26にライセンシングサービスを要求する。ライセンスサーバ20は、ライセンシングライブラリ24からサービスの要求を受信すると、データベース18からライセンストークンを検索して該トークンをライセンシングライブラリ24へと送信する。ライセンシングライブラリ24はアプリケーション26と接続しており、経路28を介してライセンスサーバ20と通信を行なう。かかる通信は、遠隔手順呼出し方式で行なわれることが望ましい。ライセンシングライブラリ24における主要なライブラリ呼出しには、アプリケーションがライセンスサーバ20に対して行なうライセンスの要求が含まれる。その他、ライセンスの更新要求と解除要求が重要なライブラリ呼出しの一部である。ライセンストークンを使用して各種のライセンシングサービスの要求を行なう方法を以下に説明する。

ライセンスアクセスモジュール(LAM)27は、ソフトウェアの販売業者が用意するものであって、ライセンストークンの復号化を目的としている。アプリケーション26は、復号化されると、ライセンシングライブラリ内のルーチンを使用してライセンストークン内のライセンシング情報を確認し、ライセンスが貸出されてよいか否かを決定する。またライセンスアクセスモジュール(LAM)27は、アプリケーションがライセンスサーバ20を介してライセンストークンをデータベース18に戻す前に、該ライセンストークンを符号化している。以下ではライセンスアクセスモジュール27をさらに詳細に説明する。

ライセンス管理ツール21は、ネットワーク管理者がソフトウェアアプリケーションの並行使用に関する管理機能を遂行する場合に使用する。ライセンス管理ツール21は、コンピュータネットワークに接続された端末であればいずれの端末でも実行可能である。ライセンス管理ツール21は、ライセンストークンをライセンスサーバ20を介してデータベース18にインストールする場合に主として使用される。ライセンス管理ツール21には次の機能が含まれる。

(1) ライセンスサーバを起動、停止する機能

(2) ライセンスサーバの制御するデータベースにアクセスする機能

(3) ライセンスの使用に関する報告を生成し、印字する機能

アプリケーション26はデータベース18を直接アクセスすることはできない。このアクセスを達成するためには、アプリケーションはライセンシングライブラリ24から経路28を介してライセンスサーバ20宛てにライセンスの要求を行なう。大部分のネットワークライセンシング方式では、ライセンシングライブラリ24とライセンスサーバ20との間に機密保護された伝送路を採用している。しかしながら本発明では、ライセンスアクセスモジュール(LAM)27、ライセンスライブラリ24および複数のライセンストークンを使用してコンピュータネットワークにおけるソフトウェアアプリケーションの不正使用を防いでいる。

再び第2図において、ライセンスサービスバインダ29は経路30を介してライセンスサーバ20と接続されている。このライセンスサービスバインダ29は、ネットワークサービスプログラム等の本分

野の公知技術によって呼出される。ライセンスサービスバインダ29は、ネットワーク上でサーバとして指定されたすべての端末の位置を検出し、いずれのサーバがいずれのアプリケーションの処理を行なっているかを記録し続ける。ライセンスサービスバインダ29は、自己の保有する可用サーバテーブルを参照して各サーバと連絡をとり、サーバの処理している製品のリストを各サーバに要求する。最後に、ライセンスサービスバインダ29は、可用ライセンスサーバテーブルの内容と製品リストを、経路32を介して結合ファイル32に書込む。第2図において、結合ファイル32は経路33を介してライセンシングライブラリ24に接続されている。アプリケーション28は結合ファイル32に問い合せを行ない、いずれのライセンスサーバが該アプリケーションのライセンスの要求を処理できるかを確認する。ライセンス生成ツール34は、ネットワーク管理者に伝送されるライセンストークンを生成するためにソフトウェア販売業者が使用する。ネットワーク管理者は、ライセンストークンを受領すると、ライセンスサーバ20を介してライセンス管理ツール21によって該トークンをデータベース18へとインストールする。

II. ライセンストークン

本発明の好ましい実施例を使用してコンピュータネットワークでライセンストークンを生成する方法を第3図を参照しながら説明する。第3図において、コンピュータネットワーク38は、ライセンス管理ツール39と1個のライセンスサーバ44とに接続されている。ライセンスサーバ44はデータベース45と通信を行なう。アプリケーション41、42、43はライセンスサーバ44からライセンシングサービスを要求している。顧客が任意のアプリケーション（例：研究開発部門用のCAD/CAMプログラム）のライセンスを購入すると、ソフトウェア販売業者はライセンス生成ツールによってライセンストークンを生成して、該トークンを顧客のネットワーク管理者に渡す。ライセンストークンは、ソフトウェアアプリケーションを使用するためのライセンスを表現する特定のビットパターンすなわちパケットである。ネットワーク管理者は、ライセンストークン46を、ライセンス管理ツール39を使用してライセンスサーバのデータベースへとインストールする。トークンリング方式で端末から端末へと転送されるトークンと異なり、本実施例のライセンストークンは、一定の時間中ライセンスサーバとライセンシングライブラリとの間でのみ転送される。かかる一定期間は、ライセンストークンがライセンスサーバから貸出される時間の長さに相当する。現在、ライセンストークンはアプリケーションに対して10秒以内に転送され、該トークンはこれを貸出したライセンスサーバへと最短時間で返送される。ライセンストークン46は、販売業者独自のフォーマットで暗号化された情報を含んでいる。かかる情報には、販売業者の識別記号、製品番号、バージョン番号、ライセンストークンに対して購入されたライセンス単位の数などが含まれる。ライセンス単位は、コンピュータネットワークに接続された端末に加重されたライセンスに対応している。たとえば、任意のソフトウェアアプリケーションを使用する場合、強力なワークステーションは平均的なパソコンより多くのライセンス単位を必要とすることになる。

ソフトウェア販売業者は、ライセンス生成ツール40を使用してライセンストークンを生成する。経路47は、顧客のサイトでライセンストークン46'がどのようにライセンス管理ツール39へと送られるかを示している。ここでは、システム管理者はライセンストークン46'をライセンストークン46としてライセンスサーバ44のライセンスデータベース45へとインストールする。経路48は、ライセンストークン46'がライセンストークン46としてライセンス管理ツール39からライセンスサーバ44へと転送される様子を示している。これにより、ライセンスサーバ44においては、トークン46に該当するアプリケーションを使用するライセンスの要求や、データベース45で示される他のアプリケーションを使用するライセンスの要求を、アプリケーション41、42、43から受け付ける用意が

完了する。

各ネットワークは複数のライセンスサーバを保有可能であり、また各ライセンスサーバはそのデータベース内に複数のソフトウェアアプリケーションに対応する複数のライセンストークンを含んでいてよいことは明らかである。再び第3図において、アプリケーションA 41が10秒未満でライセンストークン46を要求してこれを借り出すと、アプリケーションA 41がライセンスをライセンストークン46から借り出している期間中にアプリケーションB 42、C 43がこれに対する要求を行なっても、ライセンストークン46を借り出すことが出来ない。これはデータベースインタフェース19がロック機構を提供しているためである。このように、ネットワーク38でライセンスの並行使用を達成するためには、ネットワーク管理者が2個以上のライセンスサーバをインストールすることが望ましい。ライセンスサーバのクラッシュから回復するための労力を軽減するためには、システム管理者が各アプリケーションのライセンス単位を戦略的に配置された複数のライセンスサーバに分散しておくことが望ましい。たとえば、ネットワークに4個のライセンスサーバがある場合、使用頻度の高いあるアプリケーションに対して20個のライセンス単位があるとすれば、ネットワーク管理者は4個のライセンストークンの各々に5個のライセンス単位を配分することが好ましい。1個のライセンスサーバがクラッシュした場合、もしくは該サーバのライセンストークンが貸出されている場合、他の3個のライセンスサーバが他のアプリケーションに対してライセンシング処理を行なうことができる。

第4a図は、ライセンストークンを使用してライセンスを要求する方法を示している。第4a図において、ネットワーク50がアプリケーション52、54、56にそれぞれ結合されている。アプリケーション56は、ステップ59においてライセンスサーバ58からライセンストークンを要求し、該要求は受け付けられる。ステップ60において、ライセンストークンはアプリケーション56へと伝達される。続いてアプリケーション56は、ステップ61においてライセンストークンをライセンスサーバ58に送り返す。第4a図に示されるようなライセンストークンによるライセンス要求機能に加えて、ライセンシング処理の他の重要な段階でもライセンストークンが使用される。たとえば、ユーザが当初割当てられた時間を超過してアプリケーションを使用したい場合がある。この場合、第4b図に示されるように、アプリケーション66は、ライセンストークン72を使用してライセンスサーバ70に対してライセンス更新要求を発行する。同様に、第4c図に示されるように、ユーザはアプリケーションがライセンス単位を必要としなくなった時点でライセンス解除要求83を発行する。このように、ユーザはステップ85において、更新されたライセンストークンをライセンスサーバ82に戻すことによってライセンストークン84を更新する。

III. ライセンスアクセスモジュール

第2図において、ライセンスアクセスモジュール(LAM)27は、アプリケーション26とライセンシングライブラリ24とに結合されて実行可能コードを形成する。該コードは、ソフトウェア販売業者が顧客に出荷するコードである。ライセンスアクセスモジュール27は、暗号化されたライセンストークンがライセンスサーバとライセンシングライブラリ24との間で転送される時点でその復号化と符号化を行なう。このように、アプリケーションの不正使用に対する機密保護のレベルは、ライセンスアクセスモジュールの機密保護の度合いに大きく依存する。

従来のネットワークライセンシングの方法では、公共／個人キー暗号化方式を使用して重要な情報を符号化している。この方式は、ライセンスサーバが安全な環境内にある場合には効果的である。しかしながら、顧客はライセンスサーバも含めたネットワークの全端末に同じようにアクセスすることができる。ユーザがライセンスサーバの個人キーを解読すれば、ライセンシング方式の機密保護は破られてしまう。無資格のユーザがサーバの

個人キーをいったん特定すれば、ライセンスサーバに関する他のすべての重要な情報が解読されてしまう。仮にすべてのライセンスサーバが同一のキーを使用している（これがしばしば行なわれている）とすれば、すべてのライセンスサーバの保護するアプリケーションのすべての機密保護が破られてしまう。

ライセンスアクセスモジュール27は、ライセンストークンを販売業者指定のフォーマットからライセンシングライブラリ24で使用可能なフォーマットへとまず変換する。ライセンスアクセスモジュールによるこの変換は2個のモジュールによって行なわれる。第一のモジュールは、ライセンストークンを販売業者指定のフォーマットからライセンシングライブラリのフォーマットへと変換すなわち復号化する。第二のモジュールは、更新されたライセンストークンをライセンシングライブラリのフォーマットから販売業者指定のフォーマットへと変換すなわち符号化する。第二のモジュールは、ライセンシングライブラリがライセンストークン内の情報を更新するたびごとに呼出される。

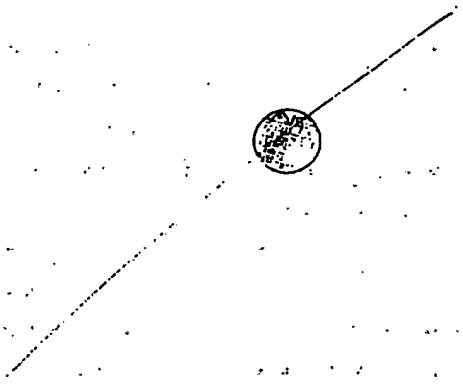
ライセンシングライブラリフォーマットに変換されたライセンストークンの受信後、ライセンシングライブラリはライセンスの正当性を確認するためのルーチンを呼出す。確認はトークン内に格納された次のようなライセンス情報を検討することによって行なわれる。

(1) フラグ
(2) 保守契約の日付
(3) ホストの名称と定義域
(4) 製品名
(5) ホストの識別番号
(6) ライセンスの通し番号
(7) ライセンスの失効の日付

以上の情報がアプリケーションの保有する情報と比較される。これらの情報が合致すれば、ライセンスの確認が完了する。確認過程の終了後、ライセンシングライブラリ内でルーチンが起動される。このルーチンは、ライセンストークン内のライセンス単位数から貸出中のライセンス単位の数を減算してライセンスを貸出す。

復号化ルーチンと符号化ルーチンとによって、ソフトウェア販売業者は、顧客のサイトに常駐するライセンスを不正使用から守るための独自の機密保護機構を構築することができる。

以下に、ライセンシングライブラリとライセンスアクセスモジュールを使用するアプリケーションであって、C言語で記述されたアプリケーションの例を示す。



```
5   #define LIC_RENEWAL_TIME (60)              /set renewal time for this session/
    #define EST_LIC_RENEWAL_TIME (LIC_RENEWAL_TIME x .9)

    NL_vendor_id NL_Vendor_Id = 1223;                  /set vendor #/
    NL_prod_num NL_Prod_num = "02"                     /set product #/
10  NL_version NL_Version = ( 12/20/88, "1.0" );       /set version id #/

    ...
        status = NL_init (vendor_id, NULL, &job_id);  /initialize license service/
        if (status != NL_NO_ERROR)      /accept job id if no error/
            {
15          fprintf (stderr, "nl_init failed - error =
            %d\n", status );                           /error message if error and
                                                       return/
            return;
            }
20      units = 3;
        code_funcs.encode_p = nl_encode;   /pointer to encode function/
        code_funcs.decode_p = nl_decode;   /pointer to decode function/
        if (signal (SIGALRM), alarm_intr ) == (void *) -1)      /set alarm if no
                                                                 error/
25          {
            perror ("Cannot set SIGALRM"); /otherwise, error message/
            return;
            }
        status = NL_request (job_id, NL_Prod_num,   /request a license/
30      &NL_Version,
        units, LIC_RENEWAL_TIME, NL_L2_SRCH,
        &code_funcs, NULL,
        &req_handle, NULL, &app_info);

        if (status != NL_NO_ERROR)                 /no error, license checked
                                                   out from license server/
35          {
            fprintf (stderr, "nl_request failed - error =
            %d\n", status);                        /otherwise, error message/
            return;
            }

40      /*
         * We got a license                        /license request successful/
         */

        alarm (EST_LIC_RENEWAL_TIME);              /set alarm for license renewal
                                                   time/
45  ...
        Application Runs                           /runs application/
    ...
        status = NL_release (req_handle);          /request to release a license/
        if (status != NL_NO_ERROR)
50          {
            fprintf (stderr, "nl_release failed - error =      /otherwise, error
```

```
                %d\n", status);                                messages/
                return;
                }

    ...
5       int
        alarm_intr ()
            {

            status = NL_confirm (req_handle,       /renew licensing unit with
            LIC_RENEWAL_TIME, NULL);                licensing server/
10      /*
         * Verify vendor private information
         */
            }

            if (status!= NL_NO_ERROR)
15          fprintf (stderr, "nl_confirm failed - error =      /otherwise, error
            %\n", status);                                     message/
                {
                    puts ("license renewed")                   /successful license
                }                                              renewal/

20
```

上記のアプリケーションの例では、コードの各種に目標の関数が付けられている。特に注目すべきコードは、code_func.encode_pとcode_func.decode_pである。encode_pとdecode_pは、ソフトウェア開発者の符号化ルーチンと復号化ルーチンにそれぞれ関係するポインタである。code_func構造のポインタを例にとると、ライセンシングライブラリは関数ポインタを使用してライセンスアクセスモジュール内の復号化ルーチンと符号化ルーチンを呼出している。3種の主要なライセンシングライブラリルーチン、すなわちライセンス獲得ルーチン(NL_request)、ライセンス解除ルーチン(NL_release)およびライセンス更新ルーチン(NL_confirm)は、上記の復号化ルーチンと符号化ルーチンを呼出す。ライセンスアクセスモジュールの例については付録Iを参照されたい。

ライセンスアクセスモジュールの実施において、ライセンスサーバはライセンストークンの受付順となるに基づくべきではない。ライセンスの付与に基づくアプリケーションの実行に先立って、アプリケー

ョンと結合されたライセンシングライブラリによってライセンストークンの確認手順が実行される。

システムの機密保護のレベルはライセンスアクセスモジュールによって決定されるため、ソフトウェア販売業者は該モジュールを任意の難易度で生成することができる。とりわけ、ソフトウェア販売業者は、暗号ルーチンの一部としていずれの暗号方式でも採用することができる。機密保持機構が解読されて暗号が知られてしまった場合、ソフトウェア販売業者は、新しいライセンスアクセスモジュールを使用して製品の新しいバージョンをリリースするだけで状況を改善することができる。

本発明は、前述のように第1-4図と付録Aを参照しながら具体的に説明され、とりわけコンピュータネットワーク環境におけるソフトウェアアプリケーションの不正使用を防ぐ方法の実施について具体的に説明されたが、かかる説明は例であって本発明を限定するものではない。また本発明による方法は、コンピュータネットワーク環境で実行されるいかなるアプリケーションに用いても効果的であることは明白である。さらに、以上の説明から本発明の精神と範囲とにそむくことなくさまざまな変更や修正を実行できることは当該分野の有能な技能者によって了承されよう。

明細書の浄書(内容に変更なし)

付録A

```
/*
 * @(#)nl_lam.c 1.21 89/09/13
 * Copyright (c) 1988 by Sun Microsystems, Inc.
 */

/*****************************************************************
 * Module: nl_lam.c
 *
 * Description:
 *      Contains the license access modules. Uses XDR to keep the license
 *      token in a machine independant form. Uses the des_crypt(3) routines
 *      to encrypt/decrypt the licensing information.
 *
 * Functions:
 *      nl_decode()     - Decodes a vendor's license token
 *      nl_encode()     - Encodes a vendor's license token
 *
 * Notes:
 *
 *****************************************************************
 */
#include        <rpc/rpc.h>
#include        <string.h>
#include        <memory.h>
#include        "nl_types.h"
#include        "nl_prot_limits.h"
#include        "nl_request.h"
#include        "nl_token.h"
#include        "nl_vendor.h"
#include        "nl_license.h"
#include        "nl_errno.h"

/*****************************************************************
 * Module Local Definitions
 *****************************************************************
 */
/*
 * Warning! This could be a potential security threat, leaving a unencoded
 * token lying around in memory.
 */
static NL_license _nl_lic_tok;

/*****************************************************************
 * Function Declarations
 *****************************************************************
 */
void            nl_lam();
static int      _nl_decode();
static int      _nl_encode();

/*****************************************************************
```

```
*
* Returns:
*      NL_NO_ERROR     - Decoding succeeded
*      NL_E_DECODE_FAIL
*
* Side Effects:
*      1) Allocates memory for the the client usage entries.
*
* Notes:
*      1) If tok_p->vend_priv_p or tok_p->vend_priv_p->data_p is NULL
*         then the call doesn't want it.
*
*****************************************************************************
 */
static int
_nl_decode(en_tok_p, en_length, tok_p, clnt_entry_p)
char            *en_tok_p;
int             en_length;
NL_token        *tok_p;
NL_clnt_entry   **clnt_entry_p;
{
        NL_usage_entry  usage_entry;
        NL_clnt_entry   *tmp_clnt_entry_p;
        XDR             xdrs;
        int             i;
        int             status;
        int             vend_size;

        /*
         * Make sure the size for the encoded token is divisible by
         * xdr bytes per unit.
         */
        if (en_length != RNDUP(en_length))
        {
                return (NL_E_DECODE_FAIL);
        }

        /*
         * Init an XDR stream
         */
        xdrmem_create(&xdrs, en_tok_p, (u_int)en_length, XDR_DECODE);

        status = NL_NO_ERROR;
        if (xdr_NL_license(&xdrs, &_nl_lic_tok) == FALSE)
        {
                xdr_destroy(&xdrs);
                return (NL_E_DECODE_FAIL);
        }

        if (_nl_lic_tok.magic == MY_VENDOR_MAGIC)
        {
                tok_p->vendor_id = _nl_lic_tok.vendor_id;
                (void) strncpy(tok_p->prod_num, _nl_lic_tok.prod_num,
                                NL_PROD_NUM_SIZE);
                tok_p->vers_secs = _nl_lic_tok.vers_secs;
                memcpy(tok_p->serial_num, _nl_lic_tok.serial_num,
```



```
* Function: nl_lam
*
* Description:
*      This function returns the address of the encode and decode routines.
*
* Input:
*      conv_2_ascii_p  - storage that indicates whether to conv to ascii
*      code_func_p     - storage containe address of encode/decode routines
*
* Output:
*      conv_2_ascii_p  - returns an indication whether the token should
*                        be converted to ascii so that it may be printable.
*      code_func_p     - Contains address of encode/decode routines
*
* Returns:
*      Nothing
*
* Notes:
*      1) The reason that the conv_2_ascii_p value is returned is to
*         inform the License Production Tool if it needs to do an
*         ascii encoding on the license token. The license token must
*         be in some type of ascii form to allow it to be transmitted over
*         voice phone or to be Fax'ed to an end user.
*
*****************************************************************************
 */
void
nl_lam(conv_2_ascii_p, code_func_p)
bool_t          *conv_2_ascii_p;
NL_code_funcs   *code_func_p;
{
        if (conv_2_ascii_p != NULL)
        {
                *conv_2_ascii_p= TRUE;
        }
        code_func_p->decode_p = _nl_decode;
        code_func_p->encode_p = _nl_encode;
} /* nl_lam */

/*
*****************************************************************************
*
* Function: nl_decode
*
* Description:
*      Translates a license token from vendor specific format to a
*      client library specific format.
*
* Input:
*      en_tok_p        - Pointer to the encoded token
*      en_length       - Length of the encoded token in bytes
*
* Output:
*      tok_p           - Pointer to the token structure to fill in
*      clnt_entry_p    - Pointer to the address of the first client usage
*                        entry
*
```

```
*       Translates a license token from a client library specific format
*       to a vendor specific format. Verifies that the encrypted token
*       does not exceed the NL_MAX_EN_TOKEN_SIZE.
*
* Input:
*       tok_p           - Pointer to the token structure to read data from
*       clnt_entry_p    - Pointer to the address of the first client usage
*                         entry
*
* Output:
*       en_tok_p        - Pointer to the encoded token
*       en_length       - Length of the encoded token in bytes
*
* Returns:
*       NL_NO_ERROR     - Encoding succeeded
*       NL_E_ENCODE_FAIL
*
* Side Effects:
*       1) Frees memory for the the client usage entries.
*
* Notes:
*       1) If tok_p->vend_priv_p is NULL then the call doesn't want it.
*
*****************************************************************************
*/
static int
nl_encode(tok_p, clnt_entry_p, en_tok_p, en_length_p)
NL_token        *tok_p;
NL_clnt_entry   *clnt_entry_p;
char            *en_tok_p;
int             *en_length_p;
{
        XDR             xdrs;
        NL_usage_entry  usage_entry;
        NL_clnt_entry   *tmp_clnt_entry_p;
        int             i;
        int             tok_len1;
        int             tok_len2;
        int             status;

        status = NL_NO_ERROR;

        /*
         * Init an XDR stream
         */
        xdrmem_create(&xdrs, en_tok_p, (u_int)NL_MAX_EN_TOKEN_SIZE, XDR_ENCODE);

        /*
         * If for some reason the license token that we will encode is
         * not the same as the previous one decoded (in our module
         * local global) then we copy the information in.
         */
        if (memcmp(_nl_lic_tok.serial_num, tok_p->serial_num,
                NL_SERIAL_NUM_SIZE) != 0)
        {
                _nl_lic_tok.magic = MY_VENDOR_MAGIC;
```

明細書の浄書(内容に変更なし)

```
                NL_SERIAL_NUM_SIZE);
        tok_p->expire_secs = _nl_lic_tok.expire_secs;
        tok_p->maint_secs = _nl_lic_tok.maint_secs;
        tok_p->total_units = _nl_lic_tok.total_units;
        tok_p->avail_units = _nl_lic_tok.avail_units;
        (void) strncpy(tok_p->host, _nl_lic_tok.host, NL_HOST_NAME_SIZE);
        (void) strncpy(tok_p->domain, _nl_lic_tok.domain,
                NL_DOMAIN_NAME_SIZE);
        tok_p->hostid = _nl_lic_tok.hostid;
        tok_p->flag = _nl_lic_tok.flag;
        tok_p->num_clnts = _nl_lic_tok.num_clnts;

        /*
         * Give the use a copy of the vendor private data if they
         * want it.
         */
        vend_size = MIN(_nl_lic_tok.vend_priv_size,
                NL_VEND_PRIV_SIZE);
        if (vend_size > 0 && tok_p->vend_priv_p != NULL &&
                tok_p->vend_priv_p->data_p != NULL)
        {
                tok_p->vend_priv_p->length = vend_size;
                (void) memcpy(tok_p->vend_priv_p->data_p,
                        _nl_lic_tok.vend_priv,
                        MIN(vend_size, tok_p->vend_priv_p->length));
        }

        tmp_clnt_entry_p = (NL_clnt_entry *)
                malloc((tok_p->num_clnts+1)* sizeof(NL_clnt_entry));
        *clnt_entry_p = tmp_clnt_entry_p;
        for (i=0; i < tok_p->num_clnts; i++, tmp_clnt_entry_p++)
        {
                xdr_NL_usage_entry(&xdrs, &usage_entry);
                tmp_clnt_entry_p->req_handle = usage_entry.req_handle;
                tmp_clnt_entry_p->units = usage_entry.units;
                tmp_clnt_entry_p->renew_secs = usage_entry.renew_secs;
                tmp_clnt_entry_p->clnt_secs = usage_entry.clnt_secs;
                tmp_clnt_entry_p->pid = usage_entry.pid;
        }
        }
        else
        {
                status = NL_E_DECODE_FAIL;
        }

        xdr_destroy(&xdrs);

        return (status);
} /* nl_decode */

/*
*****************************************************************************
*
* Function: nl_encode
*
* Description:
```

明細書の浄書(内容に変更なし)

```
    _nl_lic_tok.vendor_id = tok_p->vendor_id;
    (void) strncpy(_nl_lic_tok.prod_num, tok_p->prod_num,
                    NL_PROD_NUM_SIZE);
    _nl_lic_tok.vers_secs = tok_p->vers_secs;
    (void) memcpy(_nl_lic_tok.serial_num,tok_p->serial_num,
            NL_SERIAL_NUM_SIZE);
    _nl_lic_tok.expire_secs = tok_p->expire_secs;
    _nl_lic_tok.maint_secs = tok_p->maint_secs;
    (void) strncpy(_nl_lic_tok.host, tok_p->host, NL_HOST_NAME_SIZE);
    (void) strncpy(_nl_lic_tok.domain, tok_p->domain,
            NL_DOMAIN_NAME_SIZE);
    _nl_lic_tok.hostid = tok_p->hostid;
    _nl_lic_tok.flag = tok_p->flag;
    if (tok_p->vend_priv_p != NULL &&
            tok_p->vend_priv_p->data_p != NULL)
    {
            _nl_lic_tok.vend_priv_size =
                    MIN(tok_p->vend_priv_p->length,
                    NL_VEND_PRIV_SIZE);
            (void) memcpy(_nl_lic_tok.vend_priv,
                    tok_p->vend_priv_p->data_p,
                    _nl_lic_tok.vend_priv_size);
    }
    else
    {
            _nl_lic_tok.vend_priv_size = 0;
            _nl_lic_tok.vend_priv[0] = NUL;
    }
}
_nl_lic_tok.total_units = tok_p->total_units;
_nl_lic_tok.avail_units = tok_p->avail_units;
_nl_lic_tok.num_clnts = tok_p->num_clnts;

if (xdr_NL_license(&xdrs, &_nl_lic_tok) == FALSE)
{
        xdr_destroy(&xdrs);
        return (NL_E_ENCODE_FAIL);
}

if (clnt_entry_p != NULL)
{
        tmp_clnt_entry_p = (NL_clnt_entry *)
                malloc((_nl_lic_tok.num_clnts+1) *
                sizeof(NL_clnt_entry));
        tmp_clnt_entry_p = clnt_entry_p;
        tok_len1 = (int)xdr_getpos(&xdrs);
        for (i=0; i < tok_p->num_clnts; i++, tmp_clnt_entry_p++)
        {
                usage_entry.req_handle = tmp_clnt_entry_p->req_handle;
                usage_entry.units = tmp_clnt_entry_p->units;
                usage_entry.renew_secs = tmp_clnt_entry_p->renew_secs;
                usage_entry.clnt_secs = tmp_clnt_entry_p->clnt_secs;
                usage_entry.pid = tmp_clnt_entry_p->pid;
                xdr_NL_usage_entry(&xdrs, &usage_entry);

                /*
```

明細書の浄書(内容に変更なし)

```
                 * Quick check here for the first pass of
                 * this loop. We find out how many bytes it
                 * took to encode 1 users entry. We then
                 * estimate the total number of bytes
                 * required and make sure that we will
                 * not exceed NL_MAX_EN_TOKEN_SIZE.
                 */
                if (i == 0)
                {
                        tok_len2 = (int)xdr_getpos(&xdrs) -
                                tok_len1;
                        tok_len2 *= tok_p->num_clnts;
                        if ((tok_len1+tok_len2) >
                                NL_MAX_EN_TOKEN_SIZE)
                        {
                                i = tok_p->num_clnts;
                                status = NL_E_ENCODE_FAIL;
                        }
                }
        }
        free ((char *)clnt_entry_p);
    }
    *en_length_p = (int)xdr_getpos(&xdrs);

    xdr_destroy(&xdrs);

    return (status);
} /* nl_encode */
```

明細書の浄書(内容に変更なし)

4. 図面の簡単な説明

第1図は本発明を使用したネットワーク環境を示す図、第2図は本発明の好ましい実施例を使用したネットワークライセンシング方式の構造を示す図、第3図はライセンストークンを本発明の好ましい実施例にインストールした様子を示す図、第4a図は本発明の好ましい実施例においてライセンストークンを使用してライセンスサーバからライセンスを要求する様子を示す図、第4b図は本発明の好ましい実施例においてライセンストークンを使用してライセンスサーバからのライセンスを更新する様子を示す図、第4c図は本発明の好ましい実施例においてライセンストークンを使用してライセンスサーバからのライセンスを解放する様子を示す図である。

21・・・・ライセンス管理ツール、24・・・・ライセンシングライブラリ、34・・・・ライセンス生成ツール、32・・・・結合ファイル。

特許出願人　サン・マイクロシステムズ・インコーポレーテッド

代理人　山川政樹

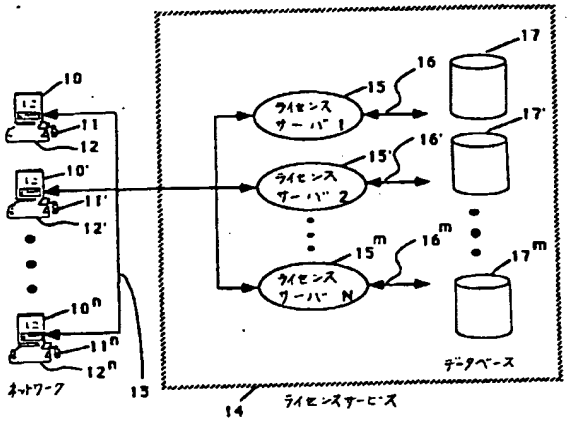


FIG. 1

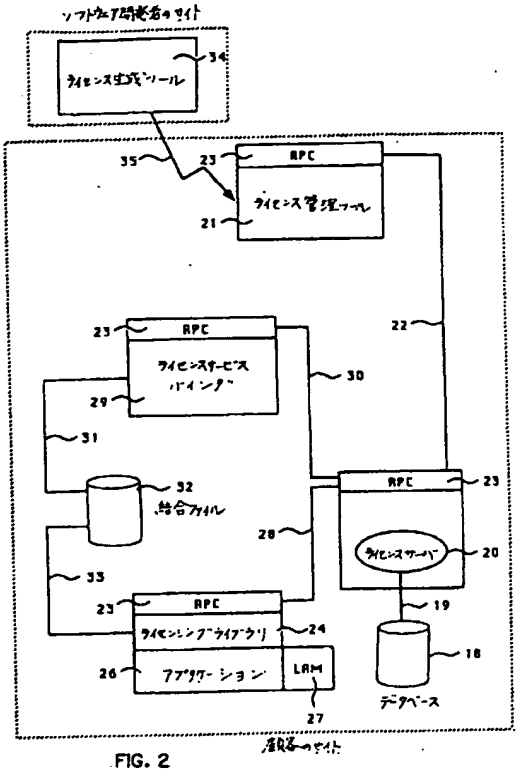


FIG. 2

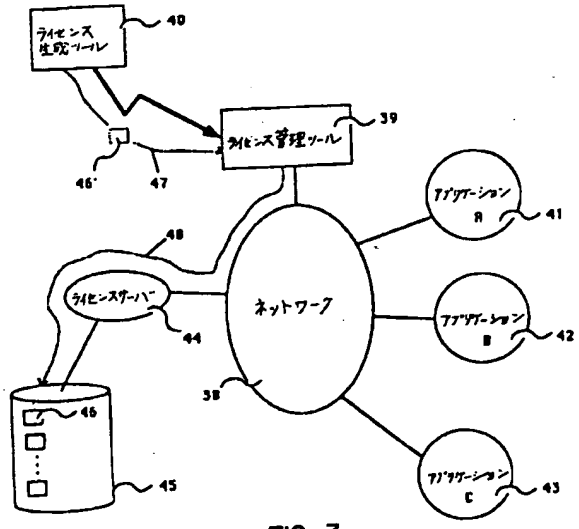


FIG. 3

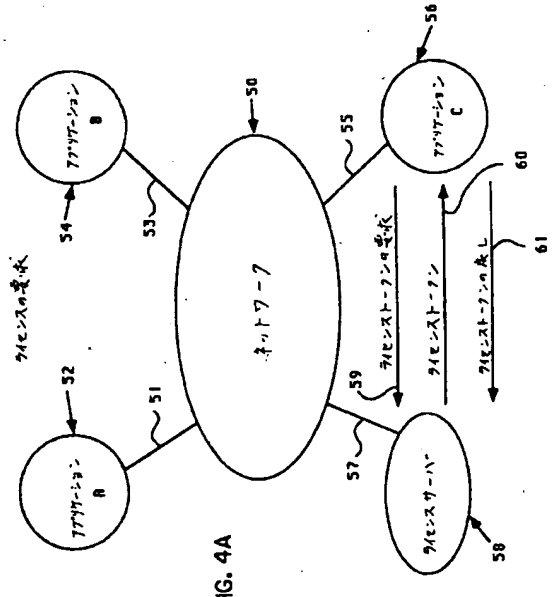


FIG. 4A

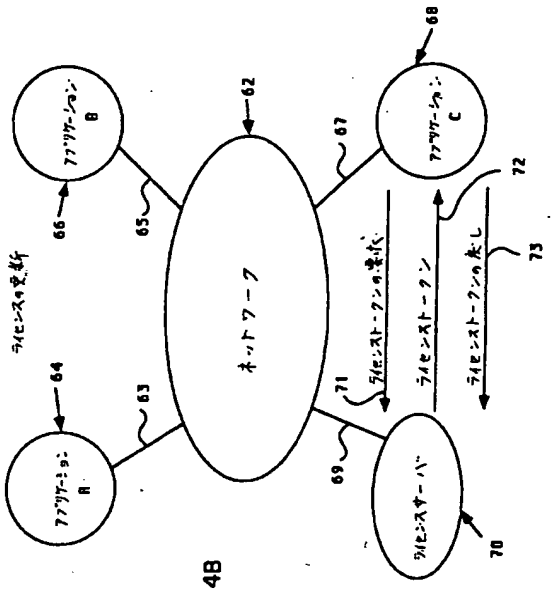


FIG. 4B

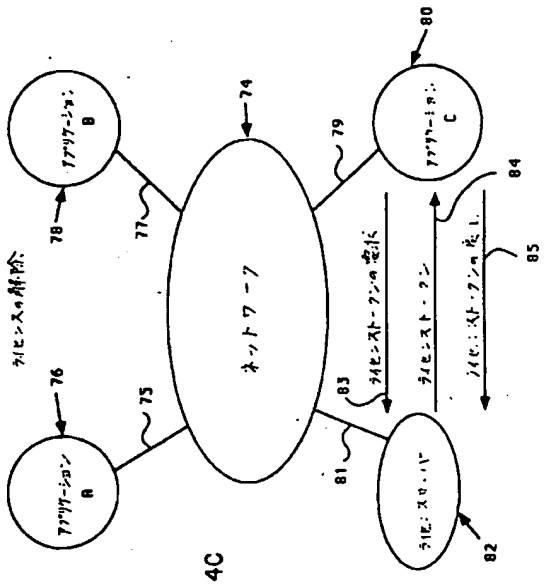


FIG. 4C

手続補正書（方式）

平成 3年 5月16日

特許庁長官殿

1．事件の表示

平成2年特許願第263242号

2．発明の名称

コンピュータネットワークシステム内での
ソフトウェアの不正使用を防ぐ方法

3．補正をする者

事件との関係　特許出願人

名称（氏名）　サン・マイクロシステムズ・
インコーポレーテッド

4．代理人

居所　東京都千代田区永田町2丁目4番2号
秀和溜池ビル8階
山川国際特許事務所内
☎（3580）0961（代表）

氏名　（6462）弁理士 山川政

5．補正命令の日付　平成 3年 4月16日

6．補正の対象

（1）明細書　41P～47P

7．補正の内容

（1）明細書の浄書（内容に変更なし）

以上

手続補正書（方式）

平成 3年 11月 7日

特許庁長官 殿

1．事件の表示　特願平2－263242号

2．発明の名称

コンピュータネットワークシステム内での
ソフトウェアの不正使用を防ぐ方法

3．補正をする者

事件との関係　特許出願人

名称　サン・マイクロシステムズ・
インコーポレーテッド

4．代理人

居所　東京都千代田区永田町2丁目4番2号
秀和溜池ビル8階
山川国際特許事務所内
電話（3580）0961（代表）

氏名（6462）弁理士 山川政

5．補正命令の日付　平成3年10月22日

6．補正の対象

明細書（第48頁）

7．補正の内容

明細書の浄書（内容に変更なし）

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☑ FADED TEXT OR DRAWING
- ☑ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☑ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.